SPECULATIVE MULTI-THREAD PROCESSING METHOD AND ITS DEVICE

Patent Number: JP2000047887
Publication date: 2000-02-18
Inventor(s): OTSU KANEMITSU; BABA KEISHIN; KOYANAGI SHIGERU
Applicant(s):: TOSHIBA CORP
Requested Patent: JP2000047887 (JP00047887)
Application Number: JP19980215945 19980730
Priority Number(s):
IPC Classification: G06F9/46 ; G06F15/16
EC Classification:
Equivalents:

Abstract

PROBLEM TO BE SOLVED: To provide a speculative multi-thread processor capable of improving the processing speed of a program.
SOLUTION: This speculative multi-thread processor is provided with plural processor parts PE0 to PEn-1 respectively executing an assigned thread, a thread manager 1 integrally managing the execution of a thread by controlling each part PE0 to PEn-1 and a memory part 2 storing a processing result, etc., by each part PE0 to PEn-1. The manager 1 assigns respectively different threads to each processor part to simultaneously and speculatively execute plural program path to improve the processing speed of the program. In addition, the manager 1 transfers data by using a register or a load store buffer in each processor part in the case of generating the necessity of transferring data between threads.

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2000－47887
（P2000－47887A）
(43)公開日　平成12年２月18日(2000.2.18)

(51)Int.Cl.'	識別記号	Ｆ Ｉ		テーマコード'（参考）
Ｇ０６Ｆ　9/46	３６０	Ｇ０６Ｆ　9/46	３６０Ｂ	５Ｂ０４５
15/16	４３０	15/16	４３０Ｚ	５Ｂ０９８

審査請求　未請求　請求項の数11　ＯＬ　（全 16 頁）

(21)出願番号　特願平10－215945

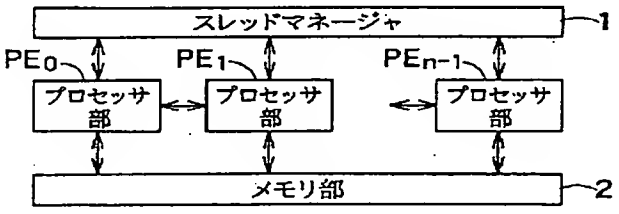
(22)出願日　平成10年７月30日(1998.7.30)

(71)出願人　000003078
株式会社東芝
神奈川県川崎市幸区堀川町72番地
(72)発明者　大　津　金　光
栃木県宇都宮市峰１－19－14　あすなろハイツ202
(72)発明者　馬　場　敬　信
栃木県宇都宮市西の宮町1176－103
(72)発明者　小　柳　滋
神奈川県川崎市幸区小向東芝町１　株式会社東芝研究開発センター内
(74)代理人　100064285
弁理士　佐藤　一雄　（外３名）

最終頁に続く

(54)【発明の名称】　投機的マルチスレッド処理方法および投機的マルチスレッド処理装置

(57)【要約】
【課題】　プログラムの処理速度を向上できる投機的マルチスレッド処理装置の提供。
【解決手段】　本発明の投機的マルチスレッド処理装置は、割り当てられたスレッドをそれぞれ実行する複数のプロセッサ部PE０～PEｎ-1と、各プロセッサ部PE０～PEｎ-1を制御してスレッドの実行を統括的に管理するスレッドマネージャ１と、各プロセッサ部PE０～PEｎ-1による処理結果等を格納するメモリ部２とを備える。スレッドマネージャ１は、各プロセッサ部にそれぞれ別個のスレッドを割り当てて、複数のプログラム経路を同時に投機的に実行させてプログラムの処理速度の向上を図る。また、スレッドマネージャ１は、スレッド間でデータ転送を行う必要が生じた場合には、各プロセッサ部内のレジスタ１２あるいはロードストアバッファ１４を用いてデータ転送を行う。



【特許請求の範囲】

【請求項１】プログラムを複数のスレッドに分割し、これらスレッドを複数のプロセッサ部に割り当てて投機的に実行させる投機的マルチスレッド処理方法において、

各プロセッサ部は、割り当てられたスレッドを投機的に実行する投機実行状態と、割り当てられたスレッドを確定的に実行する確定実行状態とを有し、

各プロセッサ部の動作を制御するスレッドマネージャにより、投機状態にある少なくとも一部のプロセッサ部を用いて、プログラム内の複数の実行経路を同時に投機的に実行させることを特徴とする投機的マルチスレッド処理方法。

【請求項２】前記スレッドマネージャは、各プロセッサ部によるスレッド実行開始、実行終了、および実行状態の切り替えと、異なるスレッド間でのデータ転送とを制御することを特徴とする請求項１に記載の投機的マルチスレッド処理方法。

【請求項３】前記スレッドマネージャは、各プロセッサ部におけるスレッドの実行状況を示すプロセッサ状態表、各プロセッサ部のスレッド割り当て状況を示すスレッド実行表、およびスレッド間の起動関係を示すスレッド関係表を有し、これらの表を用いて、各プロセッサ部が実行するスレッドを決定するとともに、各プロセッサ部によるスレッド間のデータ転送を制御することを特徴とする請求項１または２に記載の投機的マルチスレッド処理方法。

【請求項４】前記スレッドマネージャは、各プロセッサ部から通知されたスレッドの生成要求とスレッドの実行終了通知とを含む各種のスレッド制御要求コマンドを格納するFIFO型バッファを有し、このFIFO型バッファに格納された前記スレッド制御要求コマンドを順に読み出して前記プロセッサ部の動作を制御することを特徴とする請求項１～３のいずれかに記載の投機的マルチスレッド処理方法。

【請求項５】前記スレッドマネージャは、投機実行状態でのスレッドの実行が誤りであることがわかった場合には、そのスレッドの実行と、そのスレッドから起動されるすべてのスレッドの実行とを中止することを特徴とする請求項１～４のいずれかに記載の投機的マルチスレッド処理方法。

【請求項６】各プロセッサ部は、複数ビットからなるレジスタを有し、各レジスタは、異なるプロセッサ部間でレジスタ内のデータ転送を行う場合に、送信側プロセッサ部から転送されたレジスタ内のデータが受信側プロセッサ部のレジスタに格納されたか否かを示す識別ビットを有し、

受信側プロセッサ部は、前記識別ビットの値により、レジスタ内のデータ転送が終了したか否かを判断することを特徴とする請求項１～５のいずれかに記載の投機的マルチスレッド処理方法。

【請求項７】各プロセッサ部は、メモリに書き込むべきデータ、またはメモリから読み出されたデータを一時的に格納するロードストアバッファを有し、

前記ロードストアバッファは、送信側プロセッサ部内の前記ロードストアバッファから転送されたデータが受信側プロセッサ部内の前記ロードストアバッファに格納されたか否かを示す識別ビットを有し、

受信側プロセッサ部は、前記識別ビットの値により、前記ロードストアバッファのデータ転送が終了したか否かを判断することを特徴とする請求項１～６のいずれかに記載の投機的マルチスレッド処理方法。

【請求項８】各プロセッサ部が有する前記ロードストアバッファは順次に接続され、

前記ロードストアバッファは、各プロセッサ部で実行中のスレッドを起動した親スレッドに対応するプロセッサ部のＩＤ番号と、各プロセッサ部で実行中のスレッドが起動した子スレッドに対応するプロセッサ部のＩＤ番号とを格納し、投機実行状態でロード命令が通知された場合には、親スレッドに対応するロードストアバッファを参照し、投機実行状態でストア命令が通知された場合には、子スレッドに対応するロードストアバッファに内容を転送することを特徴とする請求項７に記載の投機的マルチスレッド処理方法。

【請求項９】プログラムを複数のスレッドに分割し、これらスレッドを複数の各プロセッサ部に割り当てて投機的に実行させる投機的マルチスレッド処理方法において、

各プロセッサ部は、割り当てられたスレッドを投機的に実行する投機実行状態と、割り当てられたスレッドを確定的に実行する確定実行状態とを有し、

プロセッサ部間の通信により、他のプロセッサ部によるスレッド実行開始、実行終了、および実行状態の切り替えと、異なるスレッド間でのデータ転送とを制御し、

投機実行状態にある少なくとも一部のプロセッサ部を用いて、プログラム内の複数の実行経路を同時に投機的に実行させることを特徴とする投機的マルチスレッド処理方法。

【請求項１０】プログラムを複数のスレッドに分割し、これらスレッドを複数の各プロセッサ部に割り当てて投機的に実行させる投機的マルチスレッド処理装置において、

各プロセッサ部には、割り当てられたスレッドを投機的に実行する投機実行状態と、割り当てられたスレッドを確定的に実行する確定実行状態とが設けられ、

投機実行状態にある少なくとも一部のプロセッサ部がプログラム内の複数の実行経路を同時に投機的に実行するように各プロセッサ部を制御するスレッドマネージャを備えることを特徴とする投機的マルチスレッド処理装置。

【請求項１１】プログラムを複数のスレッドに分割し、

これらスレッドを複数のプロセッサ部に割り当てて投機的に実行させる投機的マルチスレッド処理装置において、
各プロセッサ部には、割り当てられたスレッドを投機的に実行する投機実行状態と、割り当てられたスレッドを確定的に実行する確定実行状態とが設けられ、
各プロセッサ部は、他のプロセッサ部と通信を行った結果により、他のプロセッサ部のスレッド実行開始、実行終了、動作状態の切り替え、および異なるスレッド間でのデータ転送に関する情報を格納する記憶部と、
投機実行状態のときにプログラム内の複数の実行経路が同時に投機的に実行されるように、対応するプロセッサ部を制御する制御部と、を有することを特徴とする投機的マルチスレッド処理装置。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、複数のプロセッサ部を有するマルチプロセッサシステムにおいて、プログラムの実行経路を事前に予測して投機的に実行する技術に関する。

【０００２】

【従来の技術】複数のプロセッサ部を用いた並列処理の手法の中に、マルチスレッド手法と呼ばれるものがある。この手法は、一つのプログラムを複数のスレッドに分割し、各スレッドにそれぞれ異なるプロセッサ部を割り当てて並列実行させることにより、プログラムの実行を高速化するものである。ここでのスレッドは、個々の命令でもよいし、複数の命令でもよい。

【０００３】一般に、スレッド間に依存関係がある場合には、依存関係が解消するまで並列実行を抑止する方策がとられる。例えば、スレッドＢの実行がスレッドＡの実行結果に依存する場合には、スレッドＡの実行が終了するまでスレッドＢの実行が抑止される。

【０００４】これに対して、近年、スレッド間の依存関係の解消を待たずにスレッドを並列実行する投機的マルチスレッド処理方式が提案された。この方式では、プログラムの実行経路の中から１つを選択し、その選択経路（トレース）上の各スレッドについて、そのスレッドと依存関係のあるスレッドの実行終了を待たずに実行を開始する。

【０００５】図２４はあるプログラムにおけるスレッドの実行手順を示した流れ図である。また、図２５は図２４の流れ図を４台のプロセッサ部で並列実行した場合の従来方式によるスレッドの投機的な実行状況を示す図であり、図２５の縦方向は時間軸を表している。なお、各プロセッサ部の実行状態は、スレッドマネージャにより制御される。

【０００６】図２４の流れ図において、例えば、（Ａ、Ｂ、Ｃ、Ｅ）の経路を投機的に実行し、この経路が実際に実行される経路と一致したとする。この場合、まず、図２５に示すように、時刻ｔ０でプロセッサ部PE０にスレッドＡを割り当てる。プロセッサ部PE０は、スレッドＡの実行の途中（図２５の時刻ｔ１）に、スレッドＢの実行要求をスレッドマネージャに通知する。

【０００７】スレッドマネージャは、スレッドＢの実行要求（fork）を受けると、プロセッサ部PE１にスレッドＢを割り当てる。これにより、プロセッサ部PE１は、スレッドＢの実行を投機状態で開始する。

【０００８】プロセッサ部PE１は、スレッドＢの実行の途中（図２５の時刻ｔ２）に、スレッドＣの実行要求（fork）をスレッドマネージャに通知する。スレッドマネージャは、スレッドＣの実行要求（fork）を受けると、プロセッサ部PE２にスレッドＣを割り当てる。これにより、プロセッサ部PE２は、スレッドＣの実行を投機状態で開始する。

【０００９】プロセッサ部PE２は、スレッドＣの実行の途中（図２５の時刻ｔ３）に、スレッドＥの実行要求（fork）をスレッドマネージャに通知する。スレッドマネージャは、スレッドＥの実行要求（fork）を受けると、プロセッサ部PE3にスレッドＥを割り当てる。これにより、プロセッサ部PE3は、スレッドＥの実行を投機状態で開始する。

【００１０】また、プロセッサ部PE０は、スレッドＡの実行の途中（図２５の時刻ｔ３）にm、スレッドＢの実行が確定（commit）したことをスレッドマネージャに通知する。これを受けて、スレッドマネージャは、スレッドＢを実行中のプロセッサ部PE１に対して、プロセッサ部PE１が確定したことを知らせる。これを受けて、プロセッサ部PE１は、スレッドＢを確定状態に遷移させて実行する。

【００１１】プロセッサ部PE１は、スレッドＢの実行の途中（図２５の時刻ｔ４）に、スレッドＣの実行が確定（commit）したことをスレッドマネージャに通知する。これを受けて、スレッドマネージャは、スレッドＣを実行中のプロセッサ部PE２に対して、プロセッサ部PE２が確定したことを知らせる。これを受けて、プロセッサ部PE２は、スレッドＣを確定状態に遷移させて実行する。

【００１２】プロセッサ部PE２は、スレッドＣの実行の途中（図２５の時刻ｔ５）に、スレッドＤの実行が確定（commit）したことをスレッドマネージャに通知する。これを受けて、スレッドマネージャは、スレッドＥを実行中のプロセッサ部PE３に対して、プロセッサ部PE３が確定したことを知らせる。これを受けて、プロセッサ部PE３は、スレッドＥを確定状態に遷移させて実行する。

【００１３】以上のように、各プロセッサ部はスレッドを次々に起動し、各スレッドは投機状態から確定状態に遷移しながら実行される。投機的に実行したプログラム実行経路が実際の実行経路と一致する確率が高ければ、プログラムを高速に処理することが可能となり、並列処理による効率化が図れる。

【００１４】
【発明が解決しようとする課題】一方、図２６は、図２４と同様のプログラムに対して投機的な処理を行った結果、投機的に実行したプログラム経路と実際の実行経路とが一致しなかった例を示す図である。図２６の時刻ｔ０～ｔ３では、図２５と同様の処理が行われる。プロセッサ部PE１は、スレッドＢの実行の途中（図２６の時刻ｔ４）に、スレッドＣの実行が誤っていたため、処理を中止（kill）することをスレッドマネージャに通知する。これを受けて、スレッドマネージャは、スレッドＣを実行しているプロセッサ部PE２とスレッドＣにより起動されたスレッドＥを実行しているPE３とに対して、実行の中止を通知する。これを受けて、プロセッサ部PE２，PE３は、それぞれスレッドＣ，Ｅの実行を中止する。
【００１５】その後、プロセッサ部PE１は、スレッドＢの実行の途中（図２６の時刻ｔ６）に、スレッドＤの実行要求（fork）をスレッドマネージャに通知する。スレッドマネージャは、スレッドＤの実行要求（fork）を受けると、プロセッサ部PE２にスレッドＤを割り当てる。これにより、プロセッサ部PE２は、スレッドＤの実行を投機状態で開始する。
【００１６】その後、プロセッサ部PE２は、スレッドＤの実行の途中（図２６の時刻ｔ７）に、スレッドＨの実行要求（fork）をスレッドマネージャに対して行う。スレッドマネージャは、スレッドＨの実行要求（fork）を受けると、プロセッサ部PE３にスレッドＨを割り当てる。これにより、プロセッサ部PE３は、スレッドＨの実行を投機状態で開始する。
【００１７】その後、時刻ｔ８，ｔ９において、プロセッサ部PE１，PE２はそれぞれ、スレッドＤ，Ｈの実行が確定（commit）したことをスレッドマネージャに通知する。このように、投機的な実行処理を行う場合、プログラムの予測経路が的中する確率が高ければ効率よくプログラムを実行できるが、予測経路が外れる確率が高い場合には、予測処理した経路の実行を中止して、いったん後戻りして別経路を取らなければならず、プログラム全体の実行効率を向上できなくなる。
【００１８】また、従来の投機的実行方式では、複数のプログラム経路を同時に実行するような処理を行っていなかったため、単一のプログラム経路の実行確率が高くない場合には、予測が外れる可能性が高く、プログラムを高速に処理できないという問題がある。
【００１９】本発明は、このような点に鑑みてなされたものであり、その目的は、プログラムの処理速度を向上できる投機的マルチスレッド処理方法および投機的マルチスレッド処理装置を提供することにある。
【００２０】
【課題を解決するための手段】上述した課題を解決するために、本発明は、プログラムを複数のスレッドに分割し、これらスレッドを複数のプロセッサ部に割り当てて投機的に実行させる投機的マルチスレッド処理方法において、各プロセッサ部は、割り当てられたスレッドを投機的に実行する投機実行状態と、割り当てられたスレッドを確定的に実行する確定実行状態とを有し、各プロセッサ部の動作を制御するスレッドマネージャにより、投機状態にある少なくとも一部のプロセッサ部を用いて、プログラム内の複数の実行経路を同時に投機的に実行させる。
【００２１】また、本発明は、プログラムを複数のスレッドに分割し、これらスレッドを複数の各プロセッサ部に割り当てて投機的に実行させる投機的マルチスレッド処理方法において、各プロセッサ部は、割り当てられたスレッドを投機的に実行する投機実行状態と、割り当てられたスレッドを確定的に実行する確定実行状態とを有し、プロセッサ部間の通信により、他のプロセッサ部によるスレッド実行開始、実行終了、および実行状態の切り替えと、異なるスレッド間でのデータ転送とを制御し、投機実行状態にある少なくとも一部のプロセッサ部を用いて、プログラム内の複数の実行経路を同時に投機的に実行させる。
【００２２】また、本発明は、プログラムを複数のスレッドに分割し、これらスレッドを複数の各プロセッサ部に割り当てて投機的に実行させる投機的マルチスレッド処理装置において、各プロセッサ部には、割り当てられたスレッドを投機的に実行する投機実行状態と、割り当てられたスレッドを確定的に実行する確定実行状態とが設けられ、投機実行状態にある少なくとも一部のプロセッサ部がプログラム内の複数の実行経路を同時に投機的に実行するように各プロセッサ部を制御するスレッドマネージャを備える。
【００２３】また、本発明は、プログラムを複数のスレッドに分割し、これらスレッドを複数のプロセッサ部に割り当てて投機的に実行させる投機的マルチスレッド処理装置において、各プロセッサ部には、割り当てられたスレッドを投機的に実行する投機実行状態と、割り当てられたスレッドを確定的に実行する確定実行状態とが設けられ、各プロセッサ部は、他のプロセッサ部と通信を行った結果により、他のプロセッサ部のスレッド実行開始、実行終了、動作状態の切り替え、および異なるスレッド間でのデータ転送に関する情報を格納する記憶部と、投機実行状態のときにプログラム内の複数の実行経路が同時に投機的に実行されるように、対応するプロセッサ部を制御する制御部と、を有する。
【００２４】
【発明の実施の形態】以下、本発明に係る投機的マルチスレッド処理方法およびその装置について、図面を参照しながら具体的に説明する。
【００２５】（第１の実施形態）図１は本発明に係る投機的マルチスレッド処理装置の第１の実施形態の概略構

成を示すブロック図である。図１の投機的マルチスレッド処理装置は、割り当てられたスレッドをそれぞれ実行する複数のプロセッサ部PE０～PEｎ-1と、各プロセッサ部PE０～PEｎ-1を制御してスレッドの実行を管理するスレッドマネージャ１と、各プロセッサ部PE０～PEｎ-1による処理結果等を格納するメモリ部２とを備える。

【００２６】各プロセッサ部PE０～PEｎ-1は、スレッドマネージャ１から割り当てられたスレッドを実行し、スレッドの実行開始／終了、スレッドの実行に伴って生じる他のスレッドの実行、および他のスレッドとの通信や同期などの各種のスレッド制御要求をスレッドマネージャ１に通知する。スレッドマネージャ１は、各プロセッサ部PE０～PEｎ-1から要求された各種のコマンドに基づいて各プロセッサPE０～PEｎ-1を制御する。メモリ部２は、各プロセッサ部PE０～PEｎ-1からアクセス可能なように接続されている。

【００２７】図２は各プロセッサ部PE０～PEｎ-1の内部構成を示すブロック図である。図２に示すように、各プロセッサ部PE０～PEｎ-1は、プロセッサコア１１と、レジスタ１２と、キャッシュメモリ１３と、ロードストアバッファ１４とを有する。プロセッサコア１１は、例えば通常のRISCプロセッサコアからなり、スーパースカラ・プロセッサでも、VLIWプロセッサでもよい。プロセッサコア１１とレジスタ１２はスレッドマネージャ１に接続されており、各プロセッサ部PE０～PEｎ-1内のロードストアバッファ１４は相互に接続されている。

【００２８】プロセッサ部内のレジスタ１２は、図３に詳細構成を示すように、レジスタＡ１２ａとレジスタＢ１２ｂの２つからなる。レジスタＡ１２ａは通常のスレッド実行用として用いられ、レジスタＢ１２ｂは待避用として用いられる。レジスタＢ１２ｂを設ける理由は、スレッドの切り替えを高速化するためである。これらレジスタ１２ａ，１２ｂには、同期ビットが１ビットずつ設けられる。この同期ビットは、スレッドマネージャ１との同期をとるためのものであり、同期ビットの値により、スレッドマネージャ１はレジスタ内のデータ転送が行われたか否かを判断する。

【００２９】図４はスレッドマネージャ１の内部構成を示すブロック図である。図４に示すように、スレッドマネージャ１は、プロセッサ状態表２１と、スレッド実行表２２と、スレッド関係表２３と、コンテキストバッファ２４と、制御部２５と、FIFO型バッファ２６と、レジスタ間転送部２７とを有する。このうち、プロセッサ状態表２１、スレッド実行表２２およびスレッド関係表２３は、例えばＲＯＭテーブルで構成される。

【００３０】プロセッサ状態表２１は、各プロセッサ部PE０～PEｎ-1の実行状態と、実行中のスレッドのＩＤ番号と、実行コンテキストの１Ｄ番号との関係を表にしたものであり、この表により、各プロセッサ部PE０～PEｎ-1の実行状態を把握できる。各プロセッサ部PE０～PEｎ-1の実行状態には、例えば、スレッドを確定的に実行する確定実行状態、スレッドを投機的に実行する投機実行状態、およびスレッドの実行を見合わせる待機状態の３種類がある。

【００３１】スレッド実行表２２は、スレッドの実行状態と、各スレッドを実行しているプロセッサ部の番号との関係を表にしたものであり、この表により、各スレッドを実行しているプロセッサ部を把握できる。

【００３２】スレッド関係表２３は、スレッドの起動に関する情報を表にしたものであり、この表により、親スレッドと子スレッドとの関係を把握できる。

【００３３】コンテキストバッファ２４には、各プロセッサ部PE０～PEｎ-1内のレジスタ１２の値が格納される。コンテキストバッファ２４は、レジスタ１２の読み出しや書き込みを直接行うことができ、すべてのレジスタ１２のデータをコピーする場合等に用いられる。

【００３４】FIFO型バッファ２６は、各プロセッサ部PE０～PEｎ-1がスレッドマネージャ１に対して通知する各種のスレッド制御要求コマンドを格納する。スレッドマネージャ１は、FIFO型バッファ２６に格納されたコマンドを順に読み出して、コマンドに応じた制御を行う。

【００３５】図５は各プロセッサ部PE０～PEｎ-1の内部に設けられるロードストアバッファ１４（ＬＳＢ）の詳細構成を示すブロック図である。ロードストアバッファ１４は、投機実行状態のときに、スレッド実行によるメモリの整合性を保持することを目的とする。ロードストアバッファ１４には、図５に示すように、親スレッドを実行するプロセッサ部のＩＤ番号３１と、子スレッドを実行するプロセッサ部のＩＤ番号３２と、確定実行状態か投機実行状態かの実行状態を判定する判定ビット３３とが格納される。この他、ロードストアバッファ１４には、バッファエントリー情報として、ロードかストアかを識別するロードストア識別子３４と、親スレッドからの書き込み確認の同期ビット３５と、ロードストアの対象アドレス３６と、ストア時のデータ３７とが格納される。

【００３６】図６は図１の投機的マルチスレッド処理装置のプログラム実行例を示す図である。図６は、図２４のプログラムを実行する際、複数のプログラム経路を同時に実行する例を示している。

【００３７】図６の時刻ｔ０，ｔ１では、図２５の時刻ｔ０，ｔ１と同様の処理を行う。プロセッサ部PE０は、スレッドＢの実行の途中（図６の時刻ｔ２）に、スレッドＢの実行が確定（commit）したことをスレッドマネージャ１に通知する。これを受けて、スレッドマネージャ１は、スレッドＢを実行中のプロセッサ部PE１に対して、スレッドＢが確定したことを知らせる。これを受けて、プロセッサ部PE１は、スレッドＢを確定状態に遷移させて実行する。

【００３８】また、プロセッサ部PE１は、スレッドＢの

実行の途中（図６の時刻ｔ２）に、スレッドＣ，Ｄの実行要求（fork）をスレッドマネージャ１に対して行う。これを受けて、スレッドマネージャ１は、プロセッサ部PE２にスレッドＣを、プロセッサ部PE３にスレッドＤを割り当てる。これにより、プロセッサ部PE１，PE２はそれぞれ、スレッドＣ，Ｄの実行を投機状態で開始させる。

【００３９】その後、プロセッサ部PE１は、スレッドＣ，Ｄの実行の途中（図６の時刻ｔ３）に、スレッドＣの実行が確定（commit）したことと、スレッドＤの実行が誤りであったことをスレッドマネージャ１に通知する。これを受けて、スレッドマネージャ１は、プロセッサ部PE２にスレッドＣの確定を通知するとともに、プロセッサ部PE３にスレッドDの中止を通知する。これを受けて、プロセッサ部PE２はスレッドＣを確定実行状態に遷移させて実行し、プロセッサ部PE３はスレッドＤの実行を中止する。

【００４０】その後、プロセッサ部PE２は、スレッドＣの実行の途中（図６の時刻ｔ４）に、スレッドＥ，Ｆの実行要求（fork）をスレッドマネージャ１に対して行う。これを受けて、スレッドマネージャ１は、プロセッサ部PE０にスレッドＥを、プロセッサ部PE１にスレッドＦを割り当てる。これにより、プロセッサ部PE０，PE１はそれぞれ、スレッドＥ，Ｆの実行を開始させる。

【００４１】その後、プロセッサ部PE２は、スレッドＦの実行の途中（図６の時刻ｔ５）に、スレッドＦの実行が誤りであったことをスレッドマネージャ１に通知する。これを受けて、スレッドマネージャ１は、プロセッサ部PE１にスレッドＦの中止を通知する。これを受けて、プロセッサ部PE１はスレッドＦの実行を中止する。

【００４２】このように、本実施形態では、複数のプロセッサ部が異なるプログラム経路を同時に投機的に実行するため、１つのプログラム経路のみについて投機的な実行を行う場合に比べて、プログラムの実行効率を向上できる。

【００４３】図７は図６の時刻ｔ２，ｔ３，ｔ４におけるスレッドマネージャ１の内部状態を示す図である。まず、時刻ｔ２において、制御部２５は、スレッドＡ，Ｂを確定状態で実行し、スレッドＣ，Ｄを投機状態で実行することをプロセッサ状態表２１に設定する。また、制御部２５は、スレッドＡ～Ｄをそれぞれプロセッサ部PE０～PE３で実行することをスレッド実行表２２に設定する。さらに、制御部２５は、スレッドＡからスレッドＢを起動し、スレッドＢからスレッドＣを起動し、スレッドＢからスレッドＤを起動することをスレッド実行表２２に設定する。時刻ｔ３において、制御部２５は、スレッドＢを確定状態で実行し、スレッドＣを投機状態で実行することをプロセッサ状態表２１に設定する。また、制御部２５は、スレッドＢをプロセッサ部PE１で実行し、スレッドＣをプロセッサ部PE２で実行することをスレッド実行表２２に設定する。さらに、制御部２５は、スレッドＢからスレッドＣを起動することをスレッド関係表２３に設定する。

【００４４】時刻ｔ４において、制御部２５は、スレッドＥ，Ｆを投機状態で実行し、スレッドＣを確定状態で実行することをプロセッサ状態表２１に設定する。また、制御部２５は、スレッドＥをプロセッサ部PE０で、スレッドＦをプロセッサ部PE１で、スレッドＣをプロセッサ部PE２で実行することをスレッド実行表２２に設定する。また、制御部２５は、スレッドＣからスレッドＥ，Ｆを起動することをスレッド実行表２２に設定する。

【００４５】図８は図６の時刻ｔ２におけるスレッドマネージャ１の動作を模式的に示した図である。制御部２５は、図示のようなプロセッサ状態表２１とスレッド実行表２２を生成し、これらの表に基づいて、プロセッサ部PE２にスレッドＣを実行させ、プロセッサ部PE３にスレッドＤを実行させる。

【００４６】また、FIFO型バッファ２６には、プロセッサ部PE１からのスレッド制御要求コマンドが格納される。具体的には、スレッドＣの実行を確定し、スレッドＤの実行を中止する旨のスレッド制御要求コマンドがFIFO型バッファ２６に格納される。

【００４７】図９は各プロセッサ部PE０～PEn-1がスレッドマネージャ１に対して通知するコマンドの内容を説明する図である。図９の(1)に示すforkコマンドは、スレッドＴ１からスレッドＴ２の起動を要求するコマンドである。(2)に示すtermコマンドは、自己のスレッドの実行終了を通知するコマンドである。(3)に示すkillコマンドは、スレッドＴ１からスレッドＴ２を強制終了させることを要求するコマンドである。(4)に示すcommitコマンドは、スレッドＴ１からスレッドＴ２に対して、スレッドＴ２の投機実行状態から確定実行状態への遷移を通知するコマンドである。(5)に示すsendコマンドは、スレッドＴ１からスレッドＴ２に対してレジスタ１２の内容を転送するコマンドである。(6)に示すreceiveコマンドは、スレッドＴ１からの転送データをスレッドＴ２で受信するコマンドである。(7)に示すcheckコマンドは、スレッドＴ１からスレッドＴ２へのメモリ転送が完了したか否かをチェックするコマンドである。(8)に示すnotifyコマンドは、スレッドＴ１からスレッドＴ２にメモリのアドレスとデータを転送するコマンドである。

【００４８】各プロセッサ部PE０～PEn-1が図９のいずれかのコマンドをスレッドマネージャ１に対して通知すると、スレッドマネージャ１内の制御部２５は、図１０に示す処理を行う。以下、図１０のフローチャートに基づいて、制御部２５の動作を説明する。

【００４９】まず、制御部２５は、ステップＳ１に示すように、各プロセッサ部PE０～PEn-1から通知されたコ

マンドを格納しているFIFO型バッファ２６の内容を順に読み出す。読み出されたコマンドがforkコマンドの場合にはステップＳ２のfork処理を、termコマンドの場合にはステップＳ３のterm処理を、killコマンドの場合にはステップＳ４のkill処理を、commitコマンドの場合にはステップＳ５のcommit処理を、sendコマンドの場合にはステップＳ６のsend処理を、receiveコマンドの場合にはステップＳ７のreceive処理を、notifyコマンドの場合にはステップＳ８のnotify処理を、checkコマンドの場合にはステップＳ９のcheck処理を行う。

【００５０】図１１は図１０のステップＳ２のfork処理を具体的に説明するフローチャートである。まず、プロセッサ状態表２１に基づいて空きプロセッサ部を検索し（ステップＳ11）、空きプロセッサ部があれば、そのプロセッサ部のＩＤ番号を変数Ｋに代入する。

【００５１】次に、空きプロセッサ部（PE）が存在するか否かを判定し（ステップＳ12）、空きプロセッサ部が存在すれば、プロセッサ状態表２１のＫ行目に、forkコマンドの引数Ｔ２に対応するプロセッサ部を登録する（ステップＳ13）。

【００５２】次に、スレッド実行表２２のＫ行目に、引数Ｔ２に対応するプロセッサ部を登録する（ステップＳ14）。次に、引数Ｔ１，Ｔ２に対応する両プロセッサ部の起動関係をスレッド関係表２３に登録する（ステップＳ15）。次に、スレッドＴ１を実行するプロセッサ部のレジスタ１２の内容を、スレッドＴ２を実行するプロセッサ部のレジスタ１２にコピーする（ステップＳ16）。次に、スレッド実行表２２のＫ行目に設定されたプロセッサ部にスレッドＴ２の実行開始を指示する。

【００５３】一方、ステップＳ12で空きプロセッサ部がないと判定された場合には、空きプロセッサ部が見つかるまで、プロセッサ待ちバッファに登録される（ステップＳ18）。次に、スレッドＴ１を実行するプロセッサ部のレジスタ１２の内容をコンテキストバッファ２４にコピーする（ステップＳ19）。次に、スレッドＴ１からスレッドＴ２を起動することをスレッド関係表２３に設定する（ステップＳ20）。

【００５４】図１２は図１０のステップＳ３のterm処理を具体的に説明するフローチャートである。まず、スレッド実行表２２に基づいて、スレッドＴ１に対応するプロセッサ部PEを検索し、検索されたプロセッサ部に対応するＩＤ番号を変数Ｋに代入する（ステップＳ21）。

【００５５】次に、プロセッサ状態表２１の中から、変数Ｋに対応する行を削除する（ステップＳ22）。次に、スレッド実行表２２の中から、変数Ｋに対応する行を削除する（ステップＳ23）。次に、スレッド関係表２３の中から、スレッドＴ１に基づいて起動される全スレッドを削除する（ステップＳ24）。

【００５６】図１３は図１０のステップＳ４のkill処理を具体的に説明するフローチャートである。まず、スレッド実行表２２に基づいて、スレッドＴ２を実行中のプロセッサ部を検索する（ステップＳ41）。次に、プロセッサ部PE（Ｋ）に実行中止の指示を送信する（ステップＳ42）。次に、プロセッサ状態表２１のＫ行目を削除する（ステップＳ43）。次に、スレッド実行表２２のＫ行目を削除する（ステップＳ44）。次に、スレッド関係表２３のスレッドＴ１，Ｔ２を削除する（ステップＳ45）。次に、スレッド関係表２３からスレッドＴ２の子孫を検索する（ステップＳ46）。次に、スレッドＴ２の子孫に対して同様の処理を行う（ステップＳ47）。

【００５７】図１４は図１０のステップＳ５のcommit処理を具体的に説明するフローチャートである。まず、スレッド実行表２２に基づいて、スレッドＴ２を実行しているプロセッサ部を検索し、検索されたプロセッサ部に対応するＩＤ番号を変数Ｋに代入する（ステップＳ61）。次に、変数Ｋに対応するプロセッサ部に対して、このプロセッサ部が実行しているスレッドの実行が確定した旨を通知する（ステップＳ62）。次に、プロセッサ状態表２１を更新する（ステップＳ63）。すなわち、変数Ｋに対応するプロセッサ部が実行しているスレッドが確定実行状態になったことをプロセッサ状態表２１に設定する。

【００５８】図１５は図１０のステップＳ６のsend処理を具体的に説明するフローチャートである。まず、スレッドＴ１を実行しているプロセッサ部から、レジスタ１２の番号とレジスタ１２の値を受信する（ステップＳ81）。次に、スレッド実行表２２に基づいて、スレッドＴ２を実行中のプロセッサ部を検索し、検索されたプロセッサ部に対応するＩＤ番号を変数Ｋに代入する（ステップＳ82）。次に、変数Ｋに対応するプロセッサ部のレジスタ１２にデータを書き込むとともに、そのレジスタ１２の同期ビットを「１」にする。

【００５９】なお、プロセッサ部がスレッドマネージャ１に対してreceiveコマンドを通知した場合には、制御部２５は特に何も処理は行わない。

【００６０】図１６は図１０のステップＳ８のnotify処理を具体的に説明するフローチャートである。まず、スレッドＴ１を実行しているプロセッサ部から、メモリアドレスとメモリデータを受信する（ステップＳ101）。次に、スレッド実行表２２に基づいて、スレッドＴ２を実行中のプロセッサ部を検索する（ステップＳ102）。次に、検索されたプロセッサ部PE（Ｋ）に対応するプロセッサ部のロードストアバッファ１４に、ステップＳ101で受信したメモリアドレスとメモリデータを格納する（ステップＳ103）。

【００６１】なお、プロセッサ部がスレッドマネージャ１に対してcheckコマンドを通知した場合には、制御部２５は特に何も処理は行わない。

【００６２】次に、異なるスレッド間でのレジスタ１２の転送について説明する。スレッド間でデータ転送を行

うには、スレッドを同期させる必要がある。このため、本実施形態では、プロセッサ部内のレジスタ１２に同期ビットを設けてスレッドを同期させる。

【００６３】図１７はスレッド間でのレジスタ１２の転送を説明する図である。例えば、プロセッサ部PE１で実行中のスレッドＢのレジスタ１２から、プロセッサ部PE３で実行中のスレッドのレジスタ１２にデータを転送する場合について説明する。プロセッサ部PE１がスレッドＢを実行中にsendコマンドをスレッドマネージャ１に対して通知すると、スレッドマネージャ１は、通知されたsendコマンドの引数により、スレッドＢ，Ｄ間でレジスタ転送を行うことを認識する。また、スレッド実行表２２により、スレッドＤはプロセッサ部PE３が実行中であることを認識する。

【００６４】スレッドマネージャ１のレジスタ間転送部２７は、プロセッサ部PE１のレジスタ１２の内容を受信し、その受信内容をプロセッサ部PE３のレジスタ１２に転送する。

【００６５】上述したように、各プロセッサ部PE０～PEｎ-1のレジスタ１２には、同期ビットが１ビット設けられている。同期ビットの初期値は「０」であり、他のスレッドから転送されたデータがレジスタ１２に設定されたときに、対応するレジスタ１２の同期ビットは「１」に設定される。

【００６６】図１７の場合、スレッドマネージャ１がプロセッサ部PE１のレジスタ１２から転送されたデータをプロセッサ部PE３のレジスタ１２に格納したときに、プロセッサ部PE３のレジスタ１２の同期ビットは「１」に設定される。

【００６７】各プロセッサ部PE０～PEｎ-1は、スレッドＤ中のreceiveコマンドを実行するときに、同期ビットを参照する。例えば、同期ビットがすでに「１」であれば、すでにデータが転送されたと判断し、同期ビットをクリアして実行を続ける。一方、同期ビットが「０」であれば、まだスレッドＢからデータが転送されていないと判断し、同期ビットが設定されるまでプログラムの実行を中断する。

【００６８】次に、異なるスレッド間でのメモリデータの転送について説明する。ここで、メモリデータとは、メモリ部２に書き込むべきデータまたはメモリ部２から読み出したデータを指す。

【００６９】図１８はスレッド間でのメモリデータの転送を説明する図であり、プロセッサ部PE１のスレッドＢからプロセッサ部PE３のスレッドＤにメモリデータを転送する例を示している。各プロセッサ部PE０～PEｎ-1にはそれぞれロードストアバッファ１４が設けられており、このロードストアバッファ１４には、各プロセッサ部PE０～PEｎ-1がメモリ部２に書き込むべきデータとそのアドレス情報、または各プロセッサ部PE０～PEｎ-1がメモリ部２から読み出したデータとそのアドレス情報が格納される。

【００７０】具体的には、スレッドが投機実行状態のときには、メモリ部２への書き込みは行われず、ロードストアバッファ１４内に結果が保持される。スレッドが投機実行状態から確定実行状態に遷移したとき、ロードストアバッファ１４の内容がメモリに書き込まれ、スレッドの実行が無効化されたときはロードストアバッファ１４が初期化される。

【００７１】一方、メモリの読み出しを行う場合は、スレッドが確定実行状態であっても、投機実行状態であっても、実際にメモリにアクセスしてデータの読み出しを行う。

【００７２】例えば、図１８は、プロセッサ部PE1がメモリの100番地にデータ「1234」を書き込み、この番地のデータをプロセッサ部PE３が読み出す例を示している。プロセッサ部PE１は、投機実行状態のときは、メモリにデータを書き込む代わりに、ロードストアバッファ１４にデータ「1234」を書き込む。プロセッサ部PE１，PE３はそれぞれ別個にロードストアバッファ１４を有するため、プロセッサ部PE１のロードストアバッファ１４に格納されたデータは、制御部２５を介してプロセッサ部PE３のロードストアバッファ１４に転送される。

【００７３】ところで、ソースプログラムをコンパイルするときにメモリアクセス時のメモリアドレスがわかっている場合と、プログラムを実行する時点で初めてメモリアドレスがわかる場合とがある。後者は、例えば、間接アドレス等を利用している場合等である。前者と後者でメモリデータの転送手法が異なるため、以下、場合分けして説明する。

【００７４】メモリアクセス時のメモリアドレスが予めわかっている場合には、スレッドＢは、スレッドＤがメモリの100番地をアクセスすることをコンパイル時に認識する。そこで、100番地にストアすべきデータをロードストアバッファ１４に格納した直後に、格納したことを知らせるnotifyコマンドをスレッドマネージャ１に通知する。その後、100番地の内容をスレッドマネージャ１を経由して、スレッドＤを実行中のプロセッサ部PE３のロードストアバッファ１４に転送する。

【００７５】一方、スレッドＤは、100番地へのアクセスはスレッドＢがロードストアバッファ１４に格納したデータであることをコンパイル時に認識する。そこで、100番地のデータをロードする命令をスレッドマネージャ１に通知する直前にcheckコマンドを通知する。

【００７６】スレッドＤがcheckコマンドを実行する前に、スレッドＢからのデータがロードストアバッファ１４に到着していれば、checkコマンドを飛ばして次のロード命令を実行する。checkコマンドの実行前にスレッドＢからのデータがロードストアバッファ１４に到着していないときには、データの到着を待つ。データが到着したか否かを検出するため、ロードストアバッファ１４

に格納される各データには１ビットの同期ビットが設けられる。

【００７７】スレッドマネージャ１は、プロセッサ部PE１がnotifyコマンドを通知したときに、ロードストアバッファ１４の対応する番地のデータ中の同期ビットを「１」に設定する。プロセッサ部は、checkコマンドの実行時に同期ビットの値を判定して、転送データが到達したか否かを認識する。

【００７８】一方、メモリアクセスの際のメモリアドレスがプログラム実行時に初めてわかる場合には、ロードストアバッファ１４は、プロセッサ部にスレッドを割り当てて、スレッドの実行開始時にスレッドマネージャ１から親スレッドのプロセッサ部のＩＤを取得する。また、子スレッドを実行するプロセッサ部のＩＤは、スレッドがforkなどのスレッド制御命令を実行するたびにスレッドマネージャ１を経由してロードストアバッファ１４に転送される。

【００７９】ロードストアバッファ１４は、各プロセッサ部PE０～PEｎ-1の実行状態とメモリアクセスの種類により、以下のような動作を行う。

【００８０】(a)確定実行状態のプロセッサ部からメモリのロード要求があった場合

この場合、何もせずにロードストアバッファ１４よりも下のメモリ階層、すなわちメモリ部２から実際にデータをロードする。

【００８１】(b)確定実行状態のプロセッサ部からメモリのストア要求があった場合

メモリ部２にデータを書き込むと同時に、投機実行状態にある子スレッドのプロセッサ部のロードストアバッファ１４にメモリアドレスと書き込みデータを転送する。

【００８２】(c)投機実行状態のプロセッサ部からメモリのロード要求があった場合

ロードストアバッファ１４内に、該当するアドレスと一致するデータがあれば、そのデータをプロセッサ部に渡す。一致するデータがなければ、まず、親スレッドのプロセッサ部のロードストアバッファ１４を検索し、検索されたデータをプロセッサ部に渡すとともに、検索されたデータとアドレスを、自己のロードストアバッファ１４内に１エントリを割り当てて格納する。仮に、エントリの空きがなければ、空きができるまでロード処理の実行を停止する。なお、ロードストアバッファ１４のエントリは、確定実行状態への移行により解放される。投機実行状態から確定実行状態に移行すると、ロードストアバッファ１４内のすべてのデータをメモリ部２に格納した後、エントリを解放する。すなわち、ロードストアバッファ１４のエントリを越える数の投機的なメモリアクセスは行えない。また、エントリを解放すると、同期ビットはリセットされる。

【００８３】(d)投機実行状態のプロセッサ部からメモリのロード要求があった場合

ロードストアバッファ１４内に該当するデータが存在するか否かに関係なく、ロードストアバッファ１４に１エントリを割り当ててデータを格納し、かつ、子スレッドのプロセッサ部のロードストアバッファ１４にもデータを転送して格納する。もしもエントリの空きがなければ、空きができるまでロード処理の実行を停止する。この場合、同期ビットはセットしない。また、ロードストアバッファ１４は、他のプロセッサ部のロードストアバッファ１４からの要求により、次の動作を行う。

【００８４】(a)他のロードストアバッファ１４からデータのロードの問い合わせが来た場合

これは、子スレッドから親スレッドにデータを要求する場合である。自己のロードストアバッファ１４が該当データを所有している場合、すなわち、過去に投機実行状態でロードストアバッファ１４にデータを格納した場合には、そのデータを要求元に転送する。一方、自己のロードストアバッファ１４が該当データを所有していない場合、自己が確定実行状態か投機実行状態かにより、処理が異なる。確定実行状態であれば、２次キャッシュから読み込んだデータを要求元に転送する。また、投機実行状態であれば、自己の親スレッドのプロセッサ部のロードストアバッファ１４に問い合わせを行い、返ってきたデータを要求元に転送する。

【００８５】(b)他のロードストアバッファ１４からストアデータが転送されてきた場合

これは、親スレッドから子スレッドにデータを転送する場合である。自己のロードストアバッファ１４が、転送されて来たデータと同一アドレスへのアクセスをすでに行ったか否かを、ロードストアバッファ１４内の各エントリと比較して判別する。同一アドレスへのアクセスをすでに行っている場合には、プロセッサ部に対して実行のやり直しを通知する。すなわち、同時に起こったエントリにデータを書き込み、同時に同期ビットをセットする。エントリ内に同一アドレスのエントリが存在しない場合には、１エントリを割り当ててデータを格納し、同時に同期ビットをセットする。また、送られてきたデータは、エントリ内に該当アドレスがあるか否かにかかわらず、子スレッドに転送される。

【００８６】図１９は投機実行状態のロードが発生した場合の例であり、子スレッドがロード命令を実行しようとした状態を示している。図１９では、３つのスレッドＡ，Ｂ，Ｃが動作中であり、スレッドＡがスレッドＢの親、スレッドＢがスレッドＣの親である。また、スレッドＡは確定状態で実行され、スレッドＢ，Ｃは投機状態で実行されているものとする。

【００８７】スレッドＣを実行中のプロセッサ部においてロード命令が発生した場合、スレッドＣのロードストアバッファ１４には該当するエントリがないため、スレッドＣの親であるスレッドＢに問い合わせる。スレッドＡは確定実行状態なので、スレッドＡのロードストアバ

ッファ１４にエントリが存在するならばその値を、エントリが存在しなければメモリ部２をアクセスして値をスレッドＢに返す。スレッドＢは、スレッドＡから受け取った値をロードストアバッファ１４に格納するとともに、スレッドＣに返す。スレッドＣは、スレッドＢから受け取った値をロードストアバッファ１４に格納するとともにプロセッサ部に返す。

【００８８】一方、図２０は投機実行状態のストアが発生した場合の例であり、親スレッドがストア命令を実行しようとした状態を示している。スレッドＡ，Ｂ，Ｃの親子関係や実行状態は図と同じである。

【００８９】スレッドＡを実行中のプロセッサ部にてストア命令が発生した場合、スレッドＡ内のロードストアバッファ１４に１エントリを割り当ててアドレスとデータを格納するとともに、子スレッドＢを実行中のプロセッサ部にアドレスとデータを転送する。スレッドＢを実行するプロセッサ部は、自己のロードストアバッファ１４内に１エントリを割り当てて転送されてきたデータを格納するとともに、子スレッドＣにアドレスとデータを転送する。子スレッドＣを実行するプロセッサ部は、自己のロードストアバッファ１４に１エントリを割り当てて転送されてきたデータを格納する。

【００９０】（第２の実施形態）第１の実施形態は、スレッドマネージャ１により複数のプロセッサ部の動作を統括的に管理する例を説明したが、スレッドマネージャ１は必ずしも必要ではない。以下に説明する第２の実施形態は、スレッドマネージャ１を省略したものである。

【００９１】図２１は本発明に係る投機的マルチスレッド処理装置の第２の実施形態の概略構成を示すブロック図である。図２１の投機的マルチスレッド処理装置は、割り当てられたスレッドをそれぞれ実行する複数のプロセッサ部PE０～PEn-1と、各プロセッサ部PE０～PEn-1の演算結果等を格納するメモリ部２とを備えており、スレッドマネージャは備えていない。各プロセッサ部PE０～PEn-1は縦続接続されている。

【００９２】図１の装置ではスレッドマネージャがスレッドの管理を統括的に行ったが、図２１の装置では各プロセッサ部PE０～PEn-1がスレッドの管理を分散して行う。ここでは、プロセッサ部間の結合として、２本の単方向リングを想定する。スレッドの制御はリングの方向に沿って行われる。各プロセッサ部PE０～PEn-1は、単方向リングの２つの接続先それぞれに対して１つずつ不図示のフォークバッファを有する。フォークバッファ内には、スレッド制御に必要な情報が保持される。

【００９３】以下では、６台のプロセッサ部PE０～PE５により構成されるマルチプロセッサシステムを用いて、図２４の流れ図に沿って各スレッドを実行する例を説明する。図２２は６台のプロセッサ部の接続関係を示す図である。図２２に示すように、プロセッサ部PE０はプロセッサ部PE１，PE２と通信を行い、プロセッサ部PE１はプロセッサ部PE２，PE３と通信を行い、プロセッサ部PE３はプロセッサ部PE４，PE５と通信を行い、プロセッサ部PE４はプロセッサ部PE５，PE０と通信を行い、プロセッサ部PE５はプロセッサ部PE０，PE１と通信を行う。

【００９４】図２３は各プロセッサPE０～PE５の実行状態が変化する様子を示す図である。まず、図２３（ａ）に示すように、プロセッサ部PE０によりスレッドＡを実行する。その後、図２３（ｂ）に示すように、スレッドＡはスレッドＢをフォーク（fork）する。スレッドＢは、プロセッサ部PE０に隣接するプロセッサ部PE１で実行が開始される。その後、図２３（ｃ）に示すように、スレッドＢはスレッドＣ，Ｄをフォークする。スレッドＣはそれぞれ、プロセッサ部PE１と接続されているプロセッサ部PE２，PE３で実行が開始される。

【００９５】その後、図２３（ｄ）に示すように、スレッドＣはスレッドＥ，Ｆをフォークしようとするが、スレッドＣを実行しているプロセッサ部PE２に隣接するプロセッサ部PE３はすでにスレッドＤを実行中であるため、スレッドＣはスレッドＥのみをフォークする。このスレッドＥは、プロセッサ部PE４で実行が開始される。フォークできなかったスレッドＦは、プロセッサ部PE２内のフォークバッファに格納され、プロセッサ部PE３，PE４が空くのを待つ。

【００９６】その後、図２３（ｅ）に示すように、スレッドＤはスレッドＧ，Ｈをフォークしようとするが、プロセッサ部PE３に隣接するプロセッサ部PE４はスレッドＥを実行中であるため、スレッドＧのみをプロセッサ部PE５にフォークして実行を開始する。フォークできなかったスレッドＨに関しては、プロセッサ部PE３内のバッファに格納され、隣接するプロセッサ部PE４，PE５が空くのを待つ。

【００９７】その後、図２３（ｆ）に示すように、プロセッサ部PE０のスレッドＡの実行が終了し、プロセッサ部PE０は空き状態になる。その後、図２３（ｇ）に示すように、プロセッサ部PE１が実行中のスレッドＢは、スレッドＣに対してcommitコマンドを通知するとともに、スレッドＤに対してkillコマンドを通知した後、実行を終了する。一方、killコマンドを受信したプロセッサ部PE３が実行中のスレッドＤは、自己がフォークしたスレッドＧに対してkillコマンドを通知した後、実行を終了する。同様に、プロセッサ部PE５で実行中のスレッドＧは、killコマンドを受信して実行を終了する。この結果、プロセッサ部PE１，PE３，PE５が新たに空き状態になる。

【００９８】その後、図２３（ｇ）に示すように、プロセッサ部PE２のスレッドＣ内のフォークバッファに格納されていたスレッドＦが、隣接するプロセッサ部PE３が空き状態になったために実行可能になり、プロセッサ部PE３でスレッドＦの実行が開始される。

【００９９】その後、図２３（ｈ）に示すように、スレ

ッドCの実行によりスレッドEの実行が確定し、スレッドFの実行の中止が決定される。これにより、プロセッサ部PE3のスレッドFに対してkillコマンドが送信され、プロセッサ部PE3のスレッドFは実行を中止する。以上のようにして、スレッドの投機的実行が行われる。このように、第2の実施形態は、各プロセッサ部PE0～PEn-1がスレッドの管理を分散して行うため、図1に示すスレッドマネージャ1が不要となり、図1よりも装置全体の構成を簡略化できる。また、複数のプログラム経路を同時に投機的に実行してプログラムの高速処理を図る点では、第1の実施形態と同様である。

【0100】なお、第1および第2の実施形態で説明した投機的マルチスレッド処理装置は、ハードウエアで構成しても、ソフトウエアで構成してもよい。例えば、装置全体をワンチップ化すれば、高速処理が可能な高性能の中央演算処理装置(CentralProcessing Unit)を得ることができる。

【0101】

【発明の効果】以上詳細に説明したように、本発明によれば、複数のプロセッサによりプログラムを投機的に実行する場合に、プログラム内の複数の実行経路を同時に投機的に実行するようにしたため、プログラムを単一の実行経路で実行するよりもプログラムを効率的に処理でき、並列処理によるメリットを十分に生かしたマルチスレッド処理が可能となる。

【図面の簡単な説明】

【図1】投機的マルチスレッド処理装置の第1の実施形態の概略構成を示すブロック図。

【図2】各プロセッサ部の内部構成を示すブロック図。

【図3】プロセッサ部内のレジスタの詳細構成を示す図。

【図4】スレッドマネージャの内部構成を示すブロック図。

【図5】各プロセッサ部内のロードストアバッファの詳細構成を示すブロック図。

【図6】図1の装置のプログラム実行例を示す図。

【図7】図6の時刻t2～t4でのスレッドマネージャの内部状態を示す図。

【図8】図6の時刻t2でのスレッドマネージャの動作を説明する図。

【図9】各プロセッサ部がスレッドマネージャに通知するコマンドを説明する図。

【図10】スレッドマネージャ内の制御部の処理を示すフローチャート。

【図11】fork処理を具体的に説明するフローチャート。

【図12】term処理を具体的に説明するフローチャート。

【図13】kill処理を具体的に説明するフローチャート。

【図14】commit処理を具体的に説明するフローチャート。

【図15】send処理を具体的に説明するフローチャート。

【図16】notify処理を具体的に説明するフローチャート。

【図17】スレッド間でのレジスタの転送を説明する図。

【図18】スレッド間でのメモリデータの転送を説明する図。

【図19】投機実行状態のロードが発生した場合の例を示す図。

【図20】投機実行状態のストアが発生した場合の例を示す図。

【図21】投機的マルチスレッド処理装置の第3の実施形態の概略構成を示すブロック図。

【図22】6台のプロセッサ部の接続関係を示す図。

【図23】各プロセッサ部の実行状態が変化する様子を示す図。

【図24】あるプログラムにおけるスレッドの実行手順を示した流れ図。

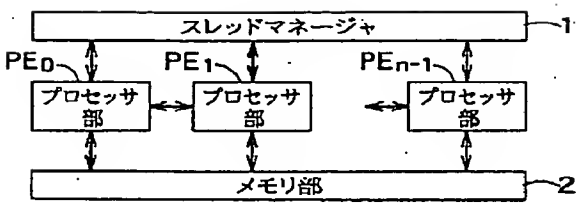
【図25】図24の流れ図を4台のプロセッサ部で並列実行した場合の従来方式によるスレッドの投機的な実行状況を示す図。

【図26】投機的に実行したプログラム経路と実際の実行経路とが一致しなかった例を示す図。

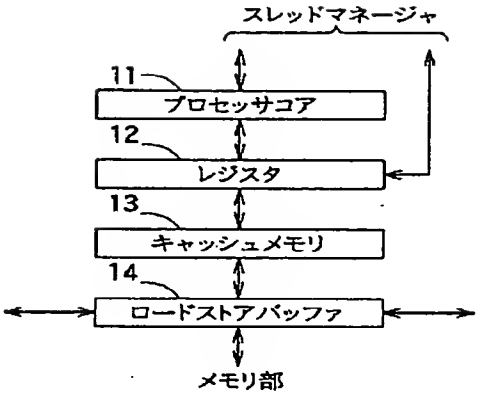
【符号の説明】

1　スレッドマネージャ
2　メモリ部
11　プロセッサコア
12　レジスタ
13　キャッシュメモリ
14　ロードストアバッファ
21　プロセッサ状態表
22　スレッド実行表
23　スレッド関係表
24　コンテキストバッファ
25　制御部
26　FIFO型バッファ
27　レジスタ間転送部

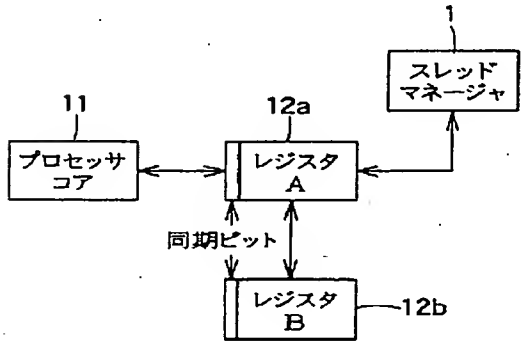
【図１】



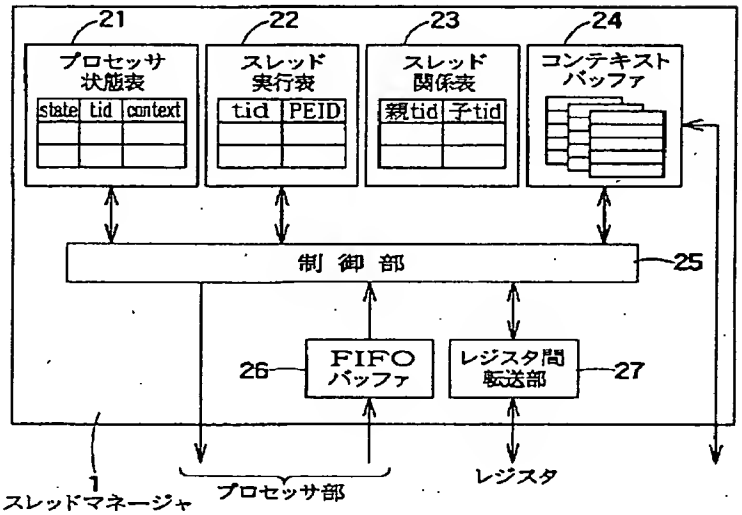
【図２】



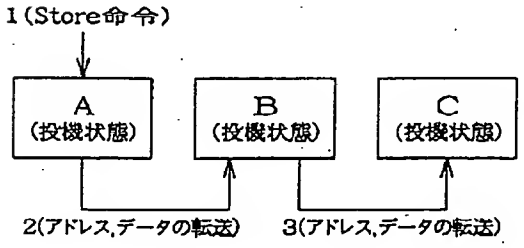
【図３】



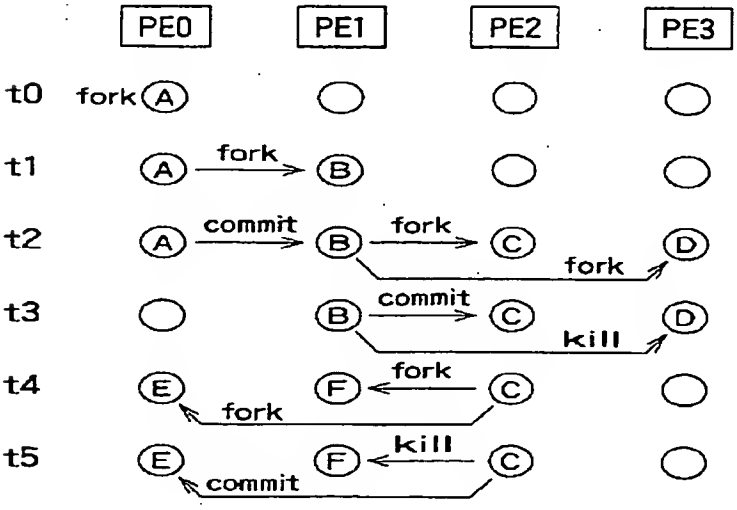
【図４】



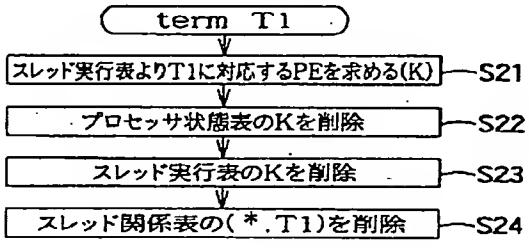
【図２０】



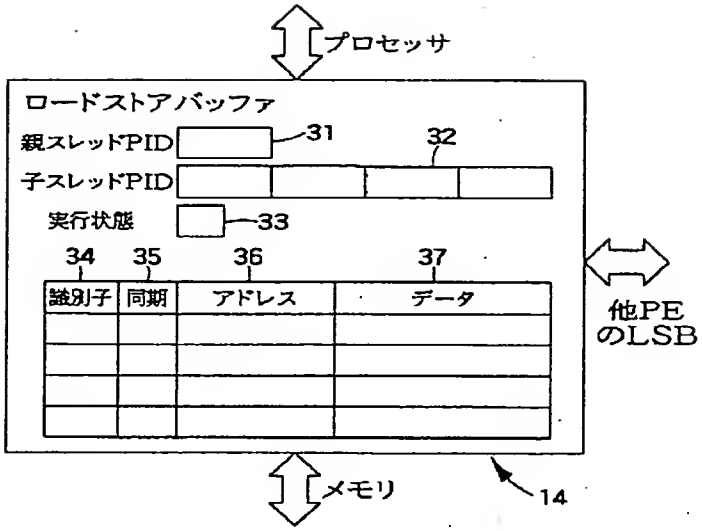
【図６】



【図１２】



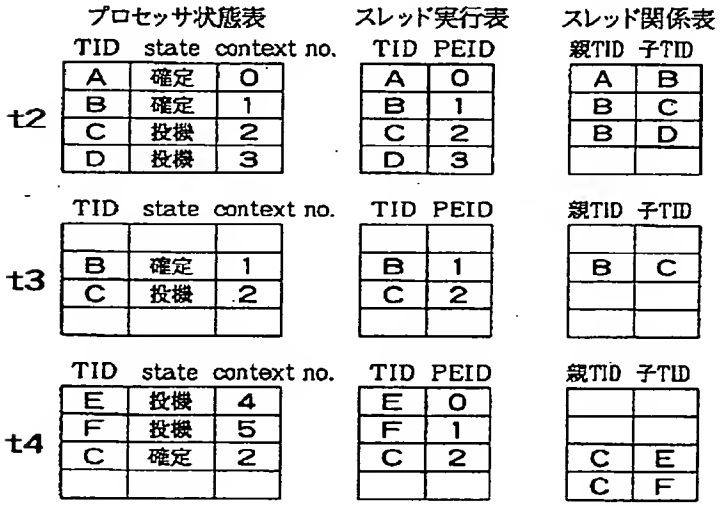
【図５】



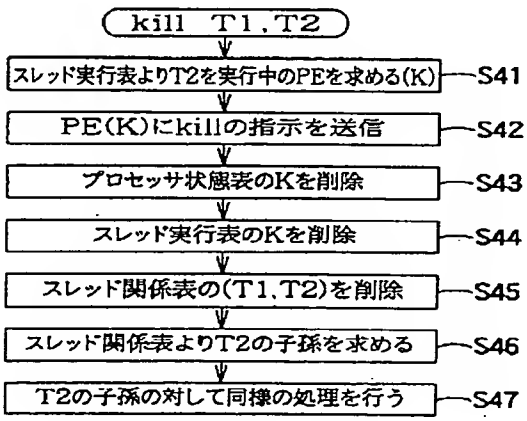
【図９】

	コマンド	機 能
(1)	fork T1,T2 addr	スレッドT1からスレッドT2の起動を要求する
(2)	term T1	自己のスレッドの実行終了を通知する
(3)	kill T1,T2	スレッドT1からスレッドT2の強制終了を要求する
(4)	commit T1,T2	スレッドT1からスレッドT2の投機実行状態から確定実行状態への遷移を通知
(5)	send T1,T2,R	スレッドT1からスレッドT2に対してレジスタの内容を転送する
(6)	receive T1,T2,R	スレッドT1から転送されたデータをスレッドT2で受信する
(7)	check T1,T2	スレッドT1からスレッドT2へのメモリ転送が完了したか否かをチェックする
(8)	notify T1,T2 addr	スレッドT1からスレッドT2にメモリのアドレスとデータを転送する

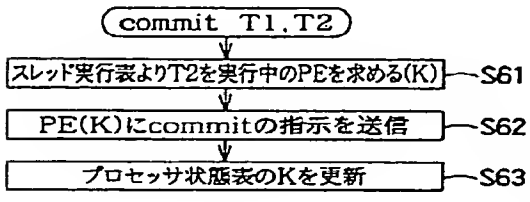
【図７】



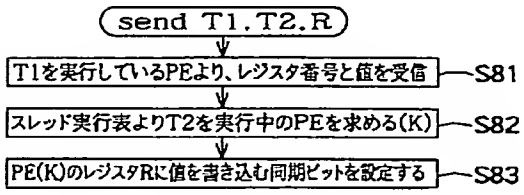
【図１３】



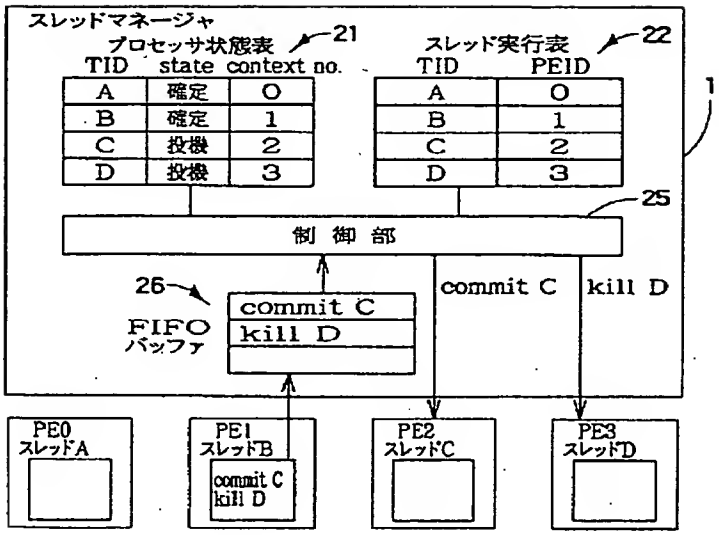
【図１４】



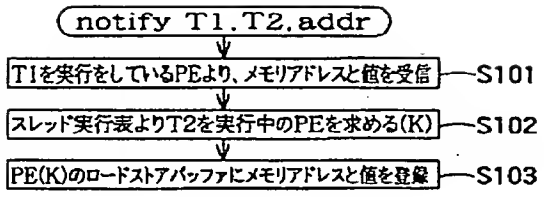
【図１５】



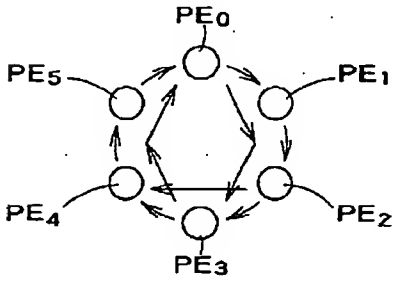
【図８】



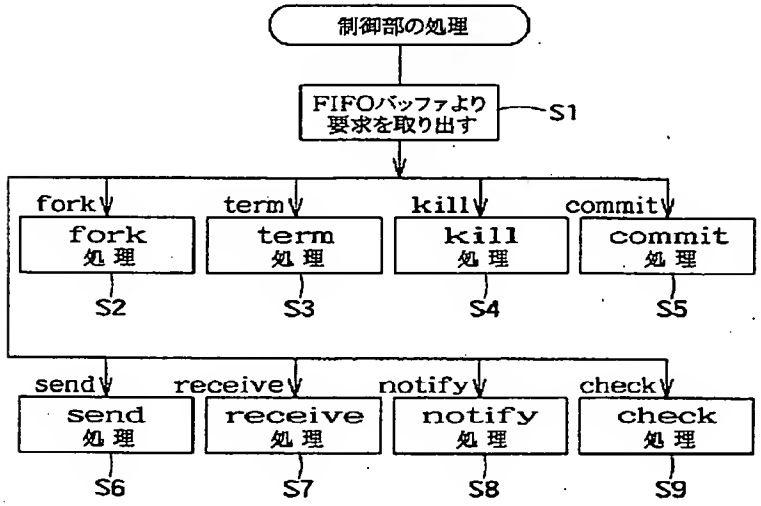
【図１６】



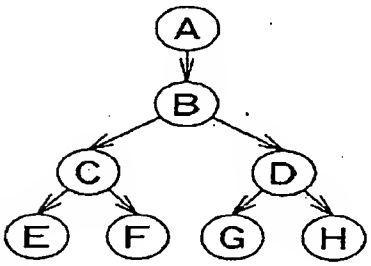
【図２２】



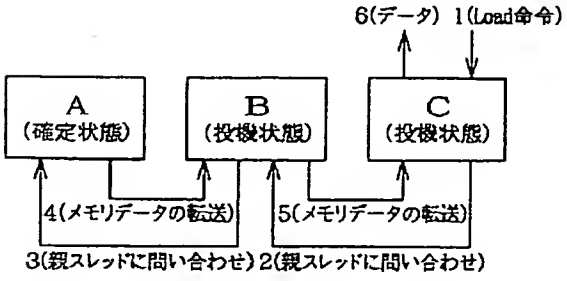
【図１０】



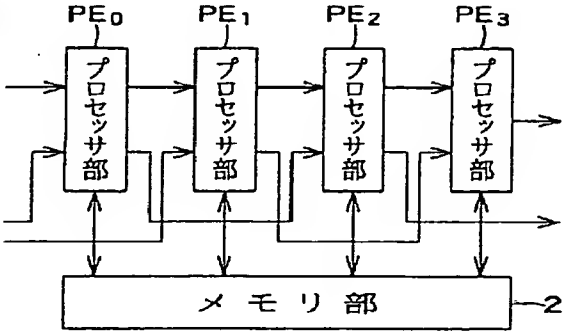
【図２４】



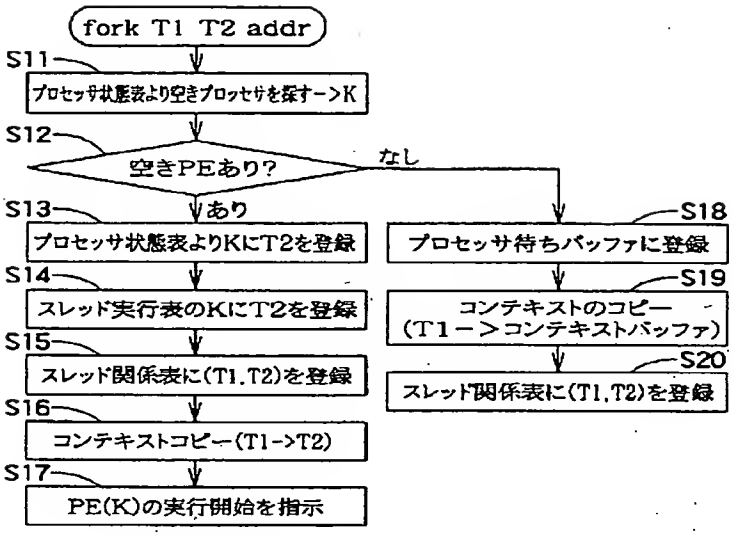
【図１９】



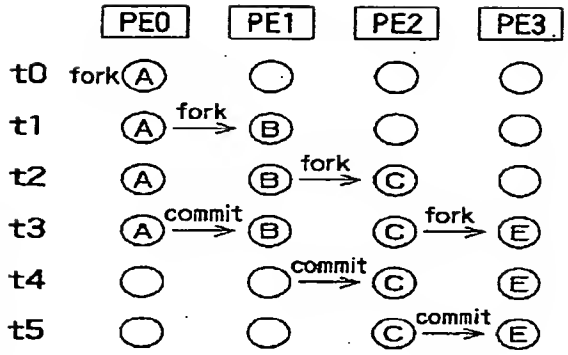
【図２１】



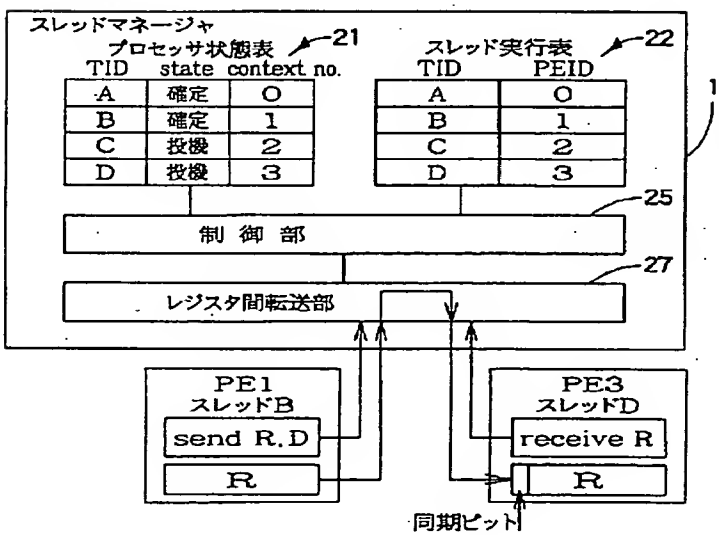
【図１１】



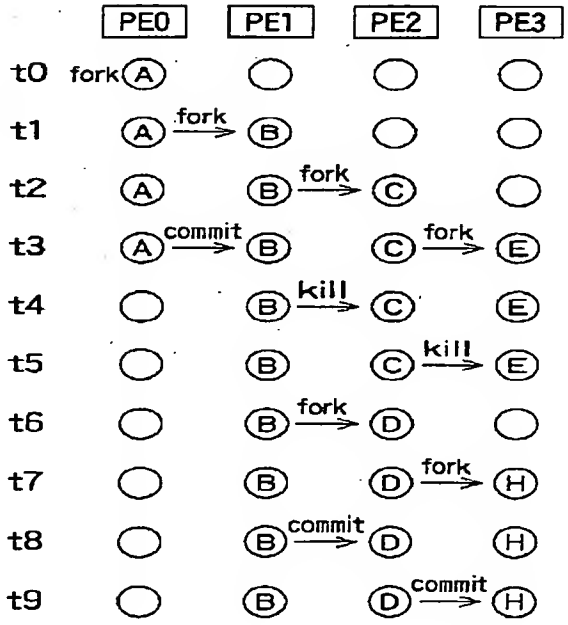
【図２５】



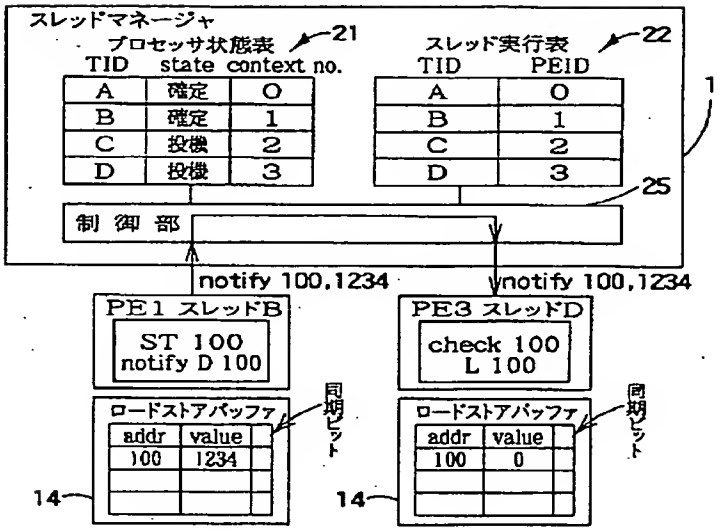
【図１７】



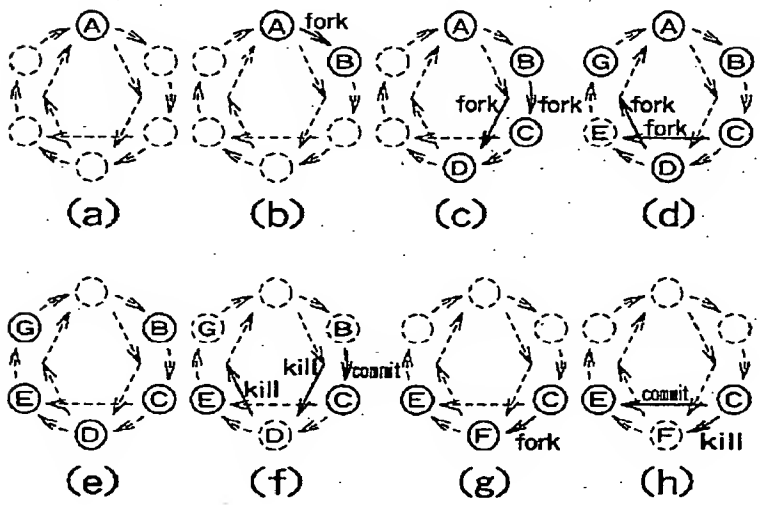
【図２６】



【図１８】



【図２３】



フロントページの続き

Ｆターム(参考)　5B045 GG11
5B098 AA10 FF07 GA05 GA08 GB09
GC10 GC15 GC16 GC19 JJ09